COMPUTER SYSTEM HAVING LOW ENERGY CONSUMPTION

BACKGROUND

1. Technical Field

The present invention relates generally to computer systems and, in particular, to a computer system having low energy consumption.

2. Background Description

In recent years, pervasive devices (PvDs) have become increasingly popular. The wireless capabilities of these devices are increasing as well as the computational complexity required by certain tasks. A major obstacle to the success of pervasive devices is energy consumption, since pervasive devices are typically powered by batteries. In general, a battery in a pervasive device may last anywhere from one hour to one year depending on the use(s) of the pervasive device. If the pervasive device is used for speech recognition and/or to support high speed wireless connectivity, it is inevitable that the energy consumption will go up significantly. The reason for this is that moderate to high performance processing units are needed to support the Million Instructions Per Second (MIPS) required by such tasks. Unfortunately, as performance goes up, energy efficiency goes down. As used herein, the phrase

"energy efficiency" is directed to joules-per-task (JPT) (and other equivalents).

Despite all the high performance and complex tasks that current pervasive devices are capable of performing, the majority of a typical pervasive device's life is spent doing simple low performance tasks such as updating the time register, periodically exchanging location information with wireless base station, sampling the ambient temperature, logging a GPS location stamp, fetching an address book entry, updating the screen buffer, detecting a keypad input, and so forth. Executing such simple tasks on a high performance processor is extremely inefficient in terms of JPT.

Thus, there have been a number of attempts to run processing units in a energy conscious manner. For example, some processor manufacturers (see, e.g., T. R. Halfhill, "Transmeta Breaks x86 Low-Power Barrier", Microprocessor Report, p. 1, Feb. 2000; and S. Liebson, "Xscale (StrongARM-2) Muscles In", Microprocessor Report, p. 1, Feb. 2000) have produced more energy efficient processing units wherein the energy consumption per cycle can be adjusted by changing supply voltage and frequency in tandem, also known as voltage scaling (see, e.g., Brodersen et al., "Low Power Digital CMOS Design", Kluwer Academic Publishers, 1995). However, there are limits to how slow these processing units

can be made to run or, rather, to how low the supply voltage can be driven. Therefore, if these processing units are primarily used to execute simple low performance tasks, then they would still dissipate significantly more energy than

5   processing units specifically designed to execute such simple tasks.

"Further, some mobile telephone and portable radio manufacturers (see, e.g., U.S. Patent No. 5,487,181, issued on Jan. 23, 1996, assigned to Ericsson GE Mobile

10   Communications Inc., Lynchburg, VA., the disclosure of which is incorporated herein by reference; and U.S. Patent No. 5,058,203, issued on Oct. 15, 1991, assigned to Fujitsu Limited, Kawasaki, Japan, the disclosure of which is incorporated herein by reference) have proposed to use

15   energy efficient application specific integrated circuits (ASIC) to offload the less energy efficient main processor in performing frequently occurring and routine functions. However, this approach for reducing energy consumption is not suitable for a more generic and more configurable

20   computing platform which is intended to offer a high degree of flexibility to the user. In a highly flexible computing system, the user may change functionality and install new features on the fly (e.g., the desktop user paradigm). The proposed conventional solutions are not flexible systems.

Rather, they are systems with specific and limited functionality.

Accordingly, there is still a need for a generic configurable computing platform having low energy consumption irrespective of system functionality demands by users, which can download new application code for evaluation and subsequent execution in the most energy efficient manner.

SUMMARY OF THE INVENTION

The problems stated above, as well as other related problems of the prior art, are solved by the present invention, a computer system having low energy consumption.

The invention provides reduced energy consumption in generic and highly configurable computing systems. Examples of some of the target computing systems that would benefit from the invention include, for example, personal digital assistants (PDAs), handheld computers, pocket computers, portable phones, laptop computers, wearable computers, and so forth. Further, the invention is particularly applicable to computing systems which are designed for a high degree of user flexibility (e.g., systems in which the user can download new applications, create his/her own applications, and/or customize the system to his/her needs). It is to be appreciated that the preceding types of computer systems

and/or computer system features are merely illustrative; the invention is applicable to many other types of computer systems having the above and/or other features, as is readily apparent to one of ordinary skill in the related

5   art.

According to a first aspect of the invention, a computer system is provided. The computer system includes at least two processing units having different energy efficiencies and adapted to at least execute tasks based

10   upon processing requirements of the tasks and a corresponding processing capability. A scheduler is adapted to schedule a given task for execution by one of the at least two processing units so as to consume a least amount of energy, and to reschedule the given task for execution by

15   an other of the at least two processing units when a determination indicates that the one of the at least two processing units is unable to accommodate execution of the given task based upon the processing requirements of the given task and the corresponding processing capability.

20   According to a second aspect of the invention, the processing requirements comprise an end time at which the given task is to be completed.

According to a third aspect of the invention, the scheduler is a function embodied within a hardware component

25   other than the at least two processing units.

According to a fourth aspect of the invention, one of the at least two processing units include the scheduler.

According to a fifth aspect of the invention, the scheduler is further adapted to intercept interrupts from the at least two processing units and peripheral devices.

According to a sixth aspect of the invention, the at least two processing units share memory space.

According to a seventh aspect of the invention, the at least two processing units share input/output space.

According to an eighth aspect of the invention, the at least two processing units share input/output space, and the scheduler and the at least two processing units share memory space.

According to a ninth aspect of the invention, the computer system further includes a task attribute store adapted to store at least some of the processing requirements of at least some of the scheduled tasks.

According to a tenth aspect of the invention, the determination is made by the scheduler.

According to an eleventh aspect of the invention, the determination is made by the one of the at least two processing units.

According to a twelfth aspect of the invention, the at least two processing units are further adapted to one of accept and reject the tasks for execution.

According to a thirteenth aspect of the invention, the processing requirements include a processing capacity required to execute the given task.

According to a fourteenth aspect of the invention, a
5   computer system is provided. The computer system includes a plurality of processing units. Each of the plurality of processing units is adapted to execute tasks thereon. At least two of the plurality of processing units have different energy efficiencies. A scheduler is adapted to
10   schedule a given task for execution by one of the plurality of processing units by querying the plurality of processing units in a partial order of descending energy efficiency to one of accept and reject the execution of the given task until one of the given task is one of accepted and executed
15   by the one of said plurality of processing units and the given task is rejected by all of the plurality of processing units.

According to a fifteenth aspect of the invention, a computer system is provided. The computer system includes
20   at least two processing units having different energy efficiencies and adapted to one of accept and reject scheduled tasks based upon processing requirements of the scheduled tasks and a corresponding processing capability. The at least two processing units are also adapted to at
25   least execute the scheduled tasks that are accepted. A

scheduler is adapted to schedule a given task for execution

by one of the at least two processing units so as to consume

a least amount of energy, and to rescheduled the given task

for execution by an other of the at least two processing

5   units when the one of the at least two processing units

rejects the execution of the given task.

According to a sixteenth aspect of the invention, a

computer system is provided. The computer system includes

a plurality of processing units. Each of the plurality of

10   processing units is adapted to execute tasks thereon. At

least two of the plurality of processing units have

different energy efficiencies. A processor attribute table

is adapted to store processing capability information for at

least some of the plurality of processors and to update the

15   processing capability information dynamically when the

processing capability information changes. A scheduler is

adapted, for a given task, to retrieve at least some of the

processing capability information from the processor

attribute table in one of a partial order and a strict order

20   of descending energy efficiency until one of the plurality

of processors is found to possess adequate processing

capability with respect to task processing requirements for

the given task. The scheduler is also adapted to schedule

the given task for execution by the one of the plurality of

25   processors.

These and other aspects, features and advantages of the present invention will become apparent from the following detailed description of preferred embodiments, which is to be read in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram illustrating a computer system 100 having low energy consumption, according to an illustrative embodiment of the invention;

FIG. 2 is a diagram illustrating a task attribute list according to an illustrative embodiment of the invention;

FIG. 3 is a diagram illustrating a processor attribute list according to an illustrative embodiment of the invention;

FIG. 4 is a flow diagram illustrating task scheduling by a scheduling unit SCH, according to an illustrative embodiment of the invention; and

FIG. 5 is a flow diagram illustrating task handling by P_j (i.e., processing units P_1 or P_2), according to an illustrative embodiment of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

The present invention is directed to a computer system having low energy consumption. It is to be understood that the present invention may be implemented in various forms of

hardware, software, firmware, special purpose processing units, or a combination thereof. Preferably, the present invention is implemented as a combination of both hardware and software. The software may be implemented as an

5   application program tangibly embodied on a program storage device. The application program may be uploaded to, and executed by, a machine comprising any suitable architecture. Preferably, the machine is implemented on a computer platform having hardware such as central processing units

10   (CPUs), a random access memory (RAM), and input/output (I/O) interface(s). The computer platform may also include an operating system and/or micro instruction code. The various processes and functions described herein may either be part of the micro instruction code or part of the application

15   program (or a combination thereof) which is executed via the operating system. In addition, various other peripheral devices may be connected to the computer platform such as an additional data storage device and a printing device.

It is to be further understood that, because some of

20   the constituent system components and method steps depicted in the accompanying Figures may be implemented in software, the actual connections between the system components (or the process steps) may differ depending upon the manner in which the present invention is programmed. Given the teachings of

25   the present invention provided herein, one of ordinary skill

in the related art will be able to contemplate these and similar implementations or configurations of the present invention.

A general description of the present invention will now be provided to introduce the reader to the concepts of the invention. Subsequently, more detailed descriptions of various aspects of the invention will be provided with respect to FIGs. 1 through 5.

To facilitate a clear understanding of the present invention, definitions of terms employed herein will now be given. The term "processing unit" is defined as a microprocessor, a microcontroller, a digital signal processor, or other device capable of executing instructions. Any combination of the preceding elements may also be employed. Also, the term "functional block" is defined as a digital or mixed signal circuit that is capable of performing one or more specific functions, but is not capable of executing instructions; moreover, the functionality of a functional block cannot be changed. An application specific integrated circuit (ASIC), a fast fourier transform (FFT), and a combinatorial circuit are considered functional blocks for the purposes of the invention. It is to be appreciated that a processing unit cannot be a functional unit, and a functional unit cannot be a processing unit. Further, the term "task" is defined as a

code portion that be independently executed concurrently with other tasks or alone. Using this definition, for example, applications, threads, interrupt service routines and the operating system kernel may be regarded as tasks.

5   Given the teachings of the invention provided herein, one of ordinary skill in the related art will contemplate these and various other implementations of the elements of the invention, while maintaining the spirit and scope thereof.

FIG. 1 is a block diagram illustrating a computer

10   system 100 having low energy consumption, according to an illustrative embodiment of the invention. The computer system 100 includes 2 processing units, P_1 and P_2 (hereinafter also generically referred to as "processing unit P_j"), and a scheduling unit, SCH. P_1 and P_2 execute

15   tasks. P_1 is the most energy efficient processing unit, but has a low computing performance. P_2 is the least energy efficient processing unit, but has a high computing performance. All interrupts from I/O space and from the two processing units are brought to SCH. MEM is the memory

20   space and I/O is the I/O space. BUS_{P1}, BUS_{P2}, BUS_{MEM}, and $BUS_{I/O}$ are the scheduler to P1 bus, scheduler to P2 bus, the memory bus, and the I/O bus, respectively. BC_{MEM} and $BC_{I/O}$ are the memory bus controller and the I/O bus controller, respectively. In a preferred embodiment of the invention,

25   the processing units P1 and P2 are executing concurrently

and each processing unit has a multitasking operating system running on it.

While SCH is implemented as a separate functional block in the illustrative embodiment of FIG. 1, it is to be

5 appreciated that the functionality of SCH may be implemented as part of the other processing units (e.g., P_1 and/or P_2), as a software program(s), or a combination thereof. For example, SCH may be implemented as a function (software, series of logic components, and so forth) within a hardware

10 component (e.g., any functional block or processing unit). Given the teachings of the invention provided herein, one of ordinary skill in the art will contemplate these and various other implementations of SCH while maintaining the spirit and scope of the invention.

15 A brief description of the operation of computer system 100 according to an illustrative embodiment of the invention will now be given. SCH intercepts all interrupt signals, resolves the interrupt to a task identification number (TID), and then negotiates execution of the task with the

20 processing units, P_1 and P_2, selecting the processing unit with the lowest energy consumption to execute and complete the task in a timely manner. In order for SCH and P_j to perform such a negotiation, certain characteristics about the task at hand should be known. These characteristics and

25 other task related information are supplied in a tasks

attribute list. FIG. 2 is a diagram illustrating a task

attribute list according to an illustrative embodiment of

the invention. It is SCH's responsibility to retrieve

certain task attributes at interrupt time. Illustrative

5   task attributes are shown and described in further detail

hereinbelow with reference to FIG. 2.

TID is the task identification number of the task at

hand. NP is the number of processing units that are able to

execute the task. $P_{TID,1}$ is the ID of the most energy

10   efficient processing unit able to execute the task. $CPS_{TID,1}$

is the worst case number of processing unit cycles needed to

run the task in a timely fashion. $ADDR_{TID,1}$ is the address of

the location of the task. $P_{TID,i}$ is the i'th most energy

efficient processing unit able to execute the task, where

15   $1<= i <= NP$. $CPS_{TID,i}$ is the worst case number of processing

unit cycles needed to run the task in a timely fashion.

$ADDR_{TID,i}$ is the address of the location of the task. $P_{TID,NP}$

is the least energy efficient processing unit of the NP

processing units able to execute the task. $CPS_{TID,NP}$ is the

20   worst case number of processing unit cycles needed to run

the task in a timely fashion. $ADDR_{TID,NP}$ is the address of

the location of the task. Note that, in principle, ADDR may

be different for each processing unit if the processing

units are not of the same architecture. For example, it is

25   not uncommon that two different processing units have

different instruction sets, and/or that one processing unit

has a 32-bit architecture and the other processing unit has

an 8-bit architecture.

It is to be emphasized that CPS represents a worst case

5   value. For example, suppose that the user launches a

DateBook application which lists a number of DateBook

entries. The corresponding task will be busy for a second

or so and then it will be in idle mode, and the DateBook

list of entries are now visible on the display. Then, the

10   user selects an entry for which he or she desires more

detailed information. The task then fetches information

from the database, displays the information on the display,

and returns to idle mode. Therefore, CPS can be described

as relating to such a task as follows: CPS represents the

15   necessary speed capability of the processing unit with

respect to the task reacting to the users input request,

fetching the information from the database, and finally

displaying the information on the display. However, for the

majority of the time, the task is in idle mode.

20   SCH will negotiate with processing units P_1 and P_2 in

the order that the processing units are listed in the

attribute list. Thus, with respect to the illustrative task

attribute list of FIG. 2, SCH will negotiate with the most

energy efficient processing unit, $P_{TID,1}$. $P_{TID,1}$ will then

25   quickly determine if it can handle the task in question

given the constraints on the required cycles per second, CPS. $P_{TID,1}$ will then either accept or reject the task. If $P_{TID,1}$ rejects the task or, if after a while, $P_{TID,1}$ determines that it will not be able to complete the task in due time,

5 $P_{TID,1}$ will then interrupt SCH. SCH will then try (negotiate with) the next processing unit in the attribute list until one of the listed processing units accepts the task or until all listed processing units have been tried.

It is to be appreciated that the task attribute list is

10 intended to enable the acceptance/rejection determination to be based upon the processing requirements of a given task to be executed and the processing capability of the processor that is to execute the given task. The processing requirements may include, for example, a minimum processing

15 capacity necessary to execute the task, a deadline time by which the task must have been executed, a specific bit-width of the data path, and so forth. The processing capability may include a currently available processing capacity, a current bit-width of the data path, and so forth. Given the

20 teachings of the invention provided herein, these and various other processing requirements and processing capabilities are readily contemplated by one of ordinary skill in the related art while maintaining the spirit and scope of the invention.

It is to be further appreciated that the actual
decision of whether a given processor can execute a given
task may be made by the processor that is queried to accept
the task or by SCH. It is preferable if such decision is

5   made based upon the processing requirements of the task and
the processing capabilities of the processor, with the
processing requirements being described in the task
attribute list, and with the processing capabilities either
accessible only by the processor or accessible also be the

10   scheduler from a commonly shared processor attribute list.
FIG. 3 is a diagram illustrating a processor attribute list
suitable for access by SCH. In FIG. 3, N is the number of
processing units, and CPSj is the current maximum available
processing capacity of processor j, where $1 <= j <= N$. The

15   processor attribute list must be dynamically updated by the
respective processors at the time the processors either
accept a new task or at the time they terminate a task, so
that the values in the attribute list represent near
instantaneous processing capabilities of the respective

20   processors. Of course, some of the processing requirements
and/or the processing capabilities may also be stored for
future reference in storage structures other than the task
attribute and processor attribute lists. Given the
teachings of the invention provided herein, one of ordinary

25   skill in the related art will contemplate these and various

other modifications to the elements of the invention while maintaining the spirit and scope thereof.

For ease of understanding, the following presumptions/conditions are imposed with respect to interrupts. SCH handles one I/O interrupt at a time. An I/O interrupt cannot preempt the handling of another I/O interrupt, nor can an I/O interrupt preempt the handling of a Processor interrupt. I/O interrupts are queued in the order in which they are received. Processor interrupts have priority over I/O interrupts.

FIG. 4 is a flow diagram illustrating task scheduling by SCH, according to an illustrative embodiment of the invention.

Upon receiving an interrupt, the category thereof is determined (step 310) If the interrupt is an I/O interrupt, then it is queued with respect to its order of arrival (step 315).

It is then determined whether the state of SCH is Idle (step 316). If not (i.e., if the state of SCH is Busy), then the SCH enters the Wait mode (step 317). In the Wait mode, SCH cannot service other I/O interrupts, although SCH does allow for queuing of I/O interrupts as described above with respect to step 315.

However, if SCH is in the Idle state, then SCH proceeds to retrieve the next interrupt in the queue (step 318),

which by default will be the just received I/O interrupt.
Next, the Busy state is entered (step 319).

The interrupt is resolved to a task attribute list
address by SCH querying an interrupt vector table (step
320). The task attributes for the task are then retrieved
(step 325). SCH then issues a request to the first
processing unit listed in the attribute list to accept or
reject the task (with task ID, TID) associated with the
current I/O interrupt (step 330). During steps 319 through
330, SCH is in the Busy state. Upon performing step 330,
SCH then enters the Wait mode (step 317). Basically, while
SCH is in the Wait mode, SCH is waiting for the current
processing unit to reply whether it is going to accept or
reject the task (with task ID, TID) associated with the
current I/O interrupt.

The current processing unit replies to SCH by
generating a Processor interrupt which, as noted above,
takes priority over I/O type interrupts. Thus, upon
determining the category of the received interrupt as a
Processor interrupt (step 310), the Processor interrupt is
then processed immediately. SCH queries processing unit
$P_{TID,i}$ for the Interrupt Function, IF_Accept (step 340).
According to the illustrative embodiment of FIG. 4, an
interrupt function IF_Accept returns a Boolean value of true
if the processing unit has accepted the task, and a Boolean

value of false if the processing unit has rejected the task. It is then determined whether the Boolean value returned by the Interrupt Function IF_Accept is true or false.

If IF_Accept is true, then SCH returns to the Idle state (step 352). Next, SCH checks if there are more I/O interrupts in the queue (step 360). If there are no more interrupts in the queue, then SCH enters into the Ready mode where it simply waits for another I/O interrupt (step 361). Otherwise, if there are more interrupts, then the method returns to step 318 where the next I/O interrupt in the queue is retrieved.

If IF_Accept is false, then the value of i is incremented by one (step 355), and the method returns to step 330 (so that SCH can issue a request to the next processing unit listed in the attribute list to accept or reject the task (with task ID, TID) associated with the current I/O interrupt).

FIG. 5 is a flow diagram illustrating task handling by P_j (i.e., processing units P_1 or P_2), according to an illustrative embodiment of the invention. For ease of understanding, certain presumptions are made as follows. A processing unit may be interrupted in two ways; by SCH and by the processor's local OS timer. An interrupt from SCH is always serviced immediately. The timer interrupt is also always serviced immediately, except if the processing unit

P_j is servicing an interrupt from SCH. The service time of a SCH interrupt is much faster than the OS master timer interrupt interval, T. OS code is stored locally, i.e., OS is not located in shared memory space, MEM.

5   Referring back to FIG. 4, it may be seen that SCH handles one I/O interrupt source at a time and that the interrupt causes SCH to negotiate with one or more processing units in a sequential fashion. Thus, SCH negotiates with one processing unit at a time and,

10   therefore, at any given time, a maximum of one interrupt from SCH is being serviced by one of the processing units. An interrupt from SCH can never be queued in a processing unit, because SCH cannot issue another interrupt until a processing unit has completely serviced the current

15   interrupt from SCH.

A brief description of task admission in P_j will now be given, followed by a more detailed description with respect to FIG. 5. When processing unit P_j receives an interrupt, processing unit P_j first determines the origin of the

20   interrupt (i.e., as coming from either SCH or by the local OS timer of the processing unit P_j). If the interrupt is the SCH interrupt, then the processing unit P_j saves the current state in local memory and services the interrupt. Thus, an interrupt from SCH may preempt any other currently

25   executing task. Upon returning from the SCH interrupt task,

the processing unit P_j resumes execution of the preempted

task. While the processing unit P_j is servicing a SCH

interrupt, interrupts are disabled. If a timer interrupt

should occur while the SCH interrupt is being serviced, then

5   the timer interrupt will not be serviced until completion of

the SCH interrupt.

Returning to FIG. 5, when processing unit P_j receives

an interrupt, it first determines the origin of the

interrupt (step 410). Upon reception of a SCH interrupt,

10   the processing unit P_j disables interrupts, and saves the

current program context (step 415). Then, the processing

unit P_j will retrieve relevant TID attributes needed for the

TID Admission check (step 420). The relevant TID attributes

include the CPS processing requirement associated with P_j.

15   The address of the task attribute table may be passed to the

processing unit by SCH (e.g., in step 325 in FIG. 4) through

a shared buffer. The TID Admission check is then performed

(step 425). The TID admission check may be performed by,

for example, the use of Equation 1 described in detail

20   hereinbelow. It is then determined whether or not the

outcome of the Admission check is either a rejection or an

acceptance of the TID (step 430). If the outcome is

acceptance, then the TID is registered, the task pointer

ADDR is inserted in the timer chain and a time slice is

25   allocated for the TID (step 435) and the method proceeds to

step 440. Otherwise, the method simply proceeds to step 440.

At step 440, SCH is notified of the outcome of the Admission check (i.e., decision of the processing unit P_j) by interrupting SCH and putting the Boolean outcome in a shared buffer that can be accessed by SCH. Then, the processing unit P_j enables interrupts again and restores task context (step 445).

With respect to allocating a time slice at step 435, the same is calculated based upon the required computing capacity (CPS) of the task (retrieved in step 420) and upon the present load of the processing unit P_j. For illustrative purposes, one way to calculate the time slice is shown in Equation 2 which is described in detail hereinbelow.

If it is determined at step 410 that the received interrupt is a timer interrupt, then the current task context is saved (step 450), the next task in the timer chain is loaded (step 475) and the local timer is programmed to interrupt according to the time slice allocated for the next task (step 480). Thus, the registered tasks are executed in a sequential fashion. When the last task in the timer chain has been executed, the timer is programmed to interrupt at the next master timer interval, T, with which the tasks are periodically loaded for execution. Note that

any registered task may be put into a dormant mode if needed (for example, if the user is not using an application) and thus be temporarily taken out of the timer chain to enable the processor to enter sleep mode as much as possible to

5   save energy.

Regarding the Admission check of step 425, consider a processing unit, P_j, which currently has K tasks executing on it. Now consider a new task, TID_{K+1}, which SCH submits for consideration to P_j. From the new tasks attribute list

10 it may be seen that TID_{K+1} needs a computing capacity of at least CPS_{TIDK+1}. Thus, for the new task, TID_{K+1}, to be admitted on P_j, the expression in Equation 1 must be satisfied, where CPS_{CLK} is the clock frequency of the processing unit, CPS_{OS} is the CPS allocated for the

15 operating system, CPS_{SCH} is the CPS allocated for interacting with SCH (i.e., the admission routine consumes part of this allocation), and CPS_{TIDk} is the CPS allocated for the k'th task currently executing on P_j. Note that all the parameters in Equation 1 are specific to P_j and that the

20 "P_j" suffix has been left out.

$$CPS_{TID_{K+1}} \leq CPS_{CLK} - CPS_{OS} - CPS_{SCH} - \sum_{k \leq K} CPS_{TID_k} \qquad (1)$$

It should be noted that determining the required CPS of a task will, in many cases, be based on experimentation by

the programmer. One way to determine the required CPS of a

task is to launch the task on the target processing unit

where there also is a so-called Throttle task running whose

CPS can be adjusted. At first, maximum CPS is made

5   available to the task, i.e., $CPS_{max} = CPS_{CLK} - CPS_{OS} - CPS_{SCH} -$

$CPS_{THROTTLE}(=0)$. Then, the programmer gradually increases the

Throttle CPS, thus reducing the available CPS of the task in

question. In this fashion, the programmer is able to

determine when the new task no longer operates in a

10   satisfactory manner.

Regarding the task time slice, consider again a

processing unit, P_j, that has K tasks executing on it. Now

consider a new task, TID_{K+1}, which SCH submits for

consideration on P_j. Further assume that each task, TID_k,

15   is executed with the master time interval, T, which is

managed by the local OS and facilitated by proper

programming of the local timer. Now, assuming P_j admits the

task, the new task, TID_{K+1}, will be allocated the time slice,

ΔT_{K+1}, according to Equation 2.

20    $$\Delta T_{K+1} = \frac{CPS_{TID_{K+1}}}{CPS_{CLK}} \bullet T \qquad\qquad (2)$$

It is to be appreciated that the invention may be

implemented such that the processing unit scheduled to

execute a task is not the processing unit which consumes the

YOR9-2000-0472US1 (8728-416)-25-

least amount of energy, but is instead the second, third,
and so on. That is, while the ultimate goal is to indeed
select the most energy efficient processing unit, other
processing units of "neighboring" energy efficiency may be
5   selected for various reasons, as readily ascertainable by
one of ordinary skill in the related art. For example, the
most energy efficient processing unit may be busy handling
another task or, for scheduling reasons, it is worthwhile to
use the second, third, or Nth most energy efficient
10   processing unit. Thus, the processing units may be queried
to accept or reject a given task for execution based on a
partial order of descending energy efficiency. The partial
order means that some processing units may be selected out
of order with respect to energy efficiency, may be excluded
15   from consideration altogether, or any combination thereof.
Given the teachings of the invention provided herein, one of
ordinary skill in the related art will contemplate these and
other variations, modifications, and extensions of task
scheduling with respect to low energy consumption, while
20   maintaining the spirit and scope of the invention.

It is to be further appreciated that the processing
units described herein may be, for example, microprocessors,
microcontrollers, digital signal processors (DSPs), or any
other devices capable of executing instructions. Moreover,
25   any combination of the preceding elements may be employed.

YOR9-2000-0472US1 (8728-416)-26-

Given the teachings of the invention provided herein, one of ordinary skill in the related art will contemplate these and various other implementations of the elements of the invention, while maintaining the spirit and scope thereof.

5   It is to be yet further appreciated that the processing units execute tasks scheduled by the scheduler and possibly other tasks which are not scheduled by the scheduler.

Although the illustrative embodiments have been described herein with reference to the accompanying drawings, it is to be understood that the present invention is not limited to those precise embodiments, and that various other changes and modifications may be affected therein by one of ordinary skill in the related art without departing from the scope or spirit of the invention. All such changes and modifications are intended to be included within the scope of the invention as defined by the appended claims.